CSS 422: Hardware and Computer Organization

Homework: Intro to Logisim

Instructor Rob Nash

## Student Name: Thuan Tran

**Number Conversions**

## Convert the following hex number to base 7; consider conversion first to decimal, and show all work. (1 pt) 0x9D7A416

9D7A4 = 4\*16^0 + 10\*16^1 + 7\*16^2 + 13\*16^3 + 9\*16^4 = 645028 (base 10)

64502810 to base 7

Left side: Quotient. Right side: Remainder

92146 6

13163 5

1880 3

268 4

38 2

5 3

0 5

Reading the remainder from bottom up => 53243567

1. Complete a truth table for the positive logic implementation of AND, OR, and NAND with two inputs A and B using true and false voltage signals. (2pt)

Positive Logic:

Low = False

High = True

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | A\*B | A+B | A NAND B |
| L | L | L | L | H |
| L | H | L | H | H |
| H | H | H | H | L |
| H | L | L | H | H |

1. Complete the truth table for the negative logic implementation of AND, OR, and NOR with two inputs A and B using true and false voltage signals. (2pt)

Negative Logic:

Low = True

High = False

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | A\*B | A+B | A NOR B |
| H | H | H | H | L |
| H | L | H | L | H |
| L | L | L | L | H |
| L | H | H | L | H |

1. Draw conclusions about: (1pt)
   1. AND in positive logic and OR in negative logic. What can we conclude here?

Looking at the result, we can conclude that AND in positive logic is the same as OR in negative logic. Just need to flip the sign

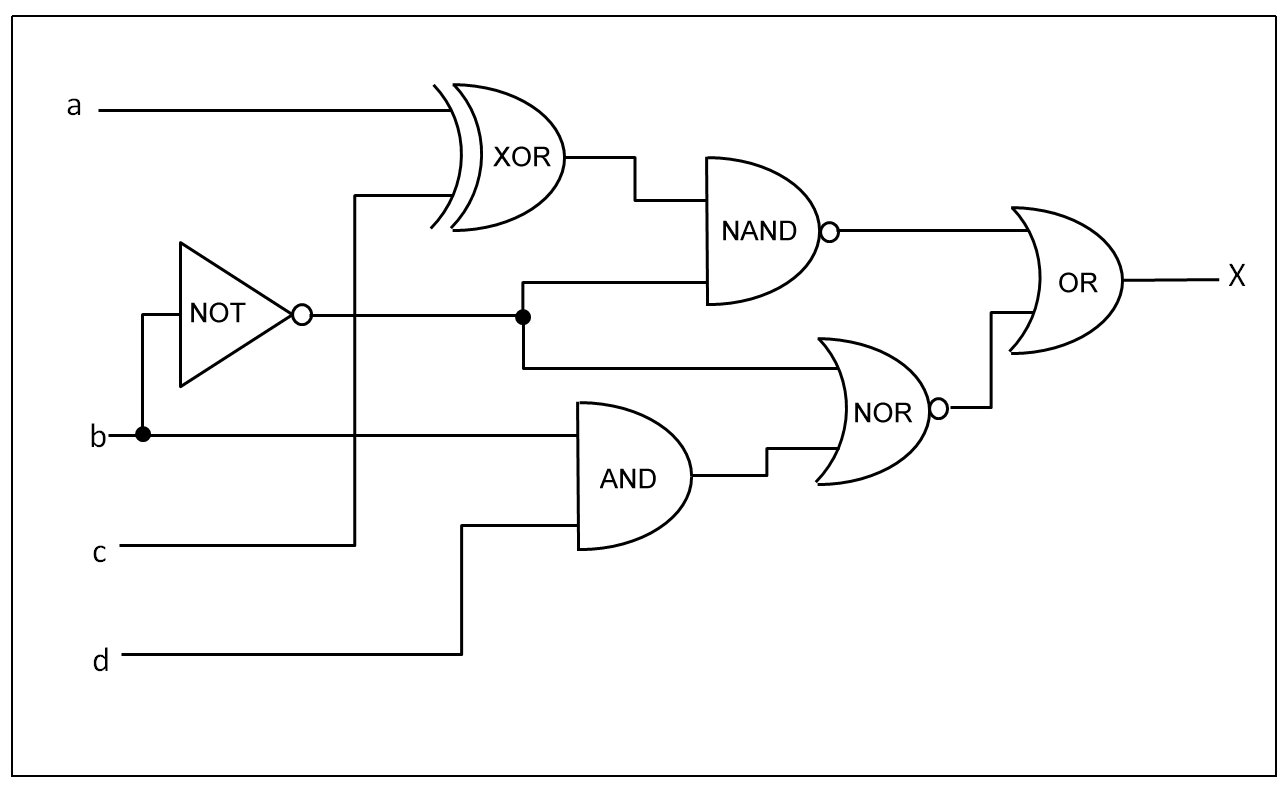
* 1. AND in positive logic and NOR in negative logic. What can we conclude here?

NOR in negative logic is the negation of AND in positive logic. (Because AND in positive logic is already OR in negative logic)

* 1. OR in negative logic and NAND in positive logic. What can we conclude here?

OR in negative logic is equivalent to the negation of NAND in positive logic. Since AND in positive logic is the same as OR in negative logic, negating NAND in positive logic will turn NAND back to AND

Given the following logic circuit, answer the questions below.



1. What is the equivalent logical Boolean equation for this circuit?

( ( A XOR C) NAND ( !B) ) + ( (BD) NOR (!B) )

1. Draw a Truth table for the four inputs {A,B,C,D} and the one output X.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | X |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

1. For the Boolean equation derived in (5), simplify this using the Boolean logic laws.

Start with the left side

( ( A XOR C) NAND ( !B) )

= ! ( ( A XOR C) . ( ! B) ) ( Transform NAND by factoring NOT outside )

= !(A XOR C ) + B (Pushing the negation inside to make AND turn into OR)

= !( (A!C) + (!AC) ) + B (Expand XOR)

Now with the right side

( (BD) NOR (!B) )

= ! (( BD) + (!B) ) (Transform NOR by factoring NOT ouside)

= ! (BD) . B (Now distribute the NOT inside)

= (! B + !D ) . B (Negate BD)

= (! BB) + (!DB) (Multiply in the B)

= False + (!DB)

= !DB

So Combining the two, now we have

!((A!C) + (!AC) ) + B + !DB

= !((A!C) + (!AC) ) + B(!D + 1) (Factor out the B)

= !((A!C) + (!AC) ) + B

= ( (!A + C) (A + !C) ) + B (Push the negation inside)

= ( (!A + C)A + (!A+C). !C) + B (Distribute them)

= ( AC + !A!C ) + B

= AC + !A!C + B (Left with OR, can remove bracket)

### Introduction to Logisim

In this section, we’ll introduce a graphical tool for designing and simulating logic circuits and systems. We’ll start by installing the software, and then we’ll walk through each tutorial.

* Download the Logisim Simulator at http://www.cburch.com/logisim/
* Find the *beginners tutorial* under the Documentation page and start at Step 0
  1. Read step 0: Orienting yourself, and summarize what you learned in this step. What did you do?

Logism has theses parts:

Explorer pane is used to contain components for our circut

Canvas is where we will draw our circuit by dragging out components from explorer panel

Attribute pane is used to display the info about our components on Canvas

Tool Bar is used to configure mode to move around : Run the simulation or drag the elements

* 1. Read step 1: Adding Gates. Explain what you did here.

Adding gates on Canvas. The Gates have dots that are the input

Adding input using the square pin. Output using the circle pin

* 1. Read step 2: Adding wires. What did you learn to do in this section?

Connecting wires together between the input pin, gates and output pins

Green wire is OK. Gray or Blue wire Not OK

* 1. Read step 3: Adding text. What did you accomplish in this section?

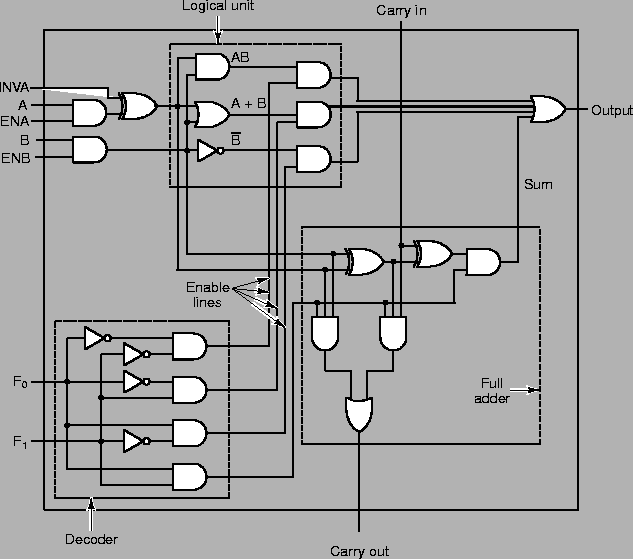
Voila, this is a XOR Circuit. This section also guided how to add text to the circuit to communicate the purpose

* 1. Read step 4: Testing your circuit. What output did you create here?

The truth table of XOR in which the output is only 1 when the 2 input are different

* 1. After completing the above exercises answering the above questions, save your circuit and submit the file along with your answers. (2 pts)

1. Simulate the following ALU with Logisim; save and submit the file. (1.5 pts)



1. Change the signals on the inputs F0 and F1, and observe how the output changes based on the A and B signals. Make a truth table for A,B, F0, F1, and output here. (1.5 pts)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | F0 | F1 | Output |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

1. Describe the four operations that correspond to the four different values for F0 and F1. (1 pt)

F0 and F1 are used to decode (decide which function to use)

So if they are both 0. Then the value that they come after the A and B gate will also be true. And if A and B is 1, then it will create an output 1. This functions likes an AND

If F0 and F1 are 0 and 1. They will connect to the AND gate after A or B. This functions like an OR

If F0 and F1 are 1 and 0, This acts like negation of B. ( Since it provide true value to the AND Gate after NOT B)

When F0 and F1 are both 1, This acts like a XOR since in which only when input A and B are different, the output is true.

1. A majority function is a combinational circuit that outputs a 1 if the input lines have more 1’s than 0’s. If there are an equal number of 1s and 0s, or more 0s than 1s, a 0 is outputted. Complete the truth table below for such a circuit that has 4 inputs {A,B,C,D} and one output X.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | X |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | !C!D | !CD | CD | C!D |
| !A!B | 0 | 0 | 0 | 0 |
| !AB | 0 | 0 | 1 | 0 |
| AB | 0 | 1 | 1 | 1 |
| A!B | 0 | 0 | 1 | 0 |

1. Derive a reduced Boolean equation for X using the Karnaugh map above.

4 loops total.

AB!CD and ABCD

!ABCD and ABCD

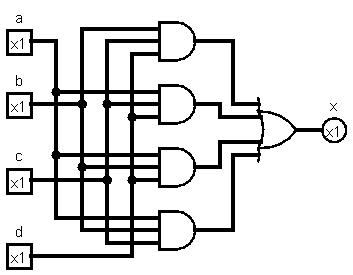
ABCD and ABC!D

A!BCD and ABCD

Only keeping the common terms, we have

ABD + BCD + ABC + ACD

1. Design this circuit using just AND, OR, and NOT gates in Logisim. Take a screenshot and attach the image to the document you submit. (6pts)



1. Could you redesign this circuit using just NAND or NOR gates? How would you do this? (1 pt)

Yes, it is possible to build this using NAND gate. We just need to remember that NAND is actually AND then NOT

So I would use double negation.

So Negate the above would be !(ABD). !(BCD) . !(ABC) . !(ACD)

Each single component is already NAND (NOT AND)

So to turn it back to normal, just need to do another negate of the above.

! ( !(ABD). !(BCD) . !(ABC) . !(ACD) ) . And voila. One big NAND gate to rule them all.